```
      begin
        temp := U[k,j1];
        U[k,j1] := c1 × temp + c2 × U[k,j];
        U[k,j] := −c2 × temp + c1 × U[k,j]
      end;
lab:  end
  end;
  for i := 1 step 1 until n do
    for j := i+1 step 1 until n do
      A[j,i] := A[i,j]
end  TRIDIAG
```

ALGORITHM 123
REAL ERROR FUNCTION, ERF(x)

MARTIN CRAWFORD AND ROBERT TECHO
Georgia Institute of Technology, Atlanta, Ga.

```
real procedure  Erf(x);  real x;
comment  Φ(x) = Erf(x) = (2/√π)∫₀ˣ e^{−u²} du can be computed
  by using the recursive relation for derivatives with Φ¹(x) =
  (2/√π)e^{−x²}, where Φ^(n)(x) = −2xΦ^(n−1)(x) − 2(n−2)Φ^(n−2)(x),
  for n = 2, 3, ··· . The Taylor's series expansions of Φ(a_k) are
  taken about k+1 points on the interval 0 < a_k ≦ x and summed
  to get Φ(x);
begin  real A, U, V, W, Y, Z, T;  integer N;
       Z := 0;  1:  if x ≠ 0 then
begin  if 0.5 < abs (x) then A := − sign (x) × 0.5
       else A := − x;
       U := V := 1.12837917 × exp(−x↑2);  Y := T := −V ×
         A;  N := 1;
2:     if abs(T) ≧ 10− 10 then
begin  N := N + 1;  W := −2 × x × V − 2 × U × (N−2);
         T := T × W × A/(V × N);
       U := V;  V := W;  Y := Y + T;  go to 2 end;
       Z := Z + Y;  x := x + A;  go to 1 end;
       Erf := Z end Erf
```

ALGORITHM 124
HANKEL FUNCTION

LUIS J. SCHAEFER
Purdue University, West Lafayette, Ind.

```
procedure  HANKEL(N,X,H);  value N,X;  integer N;
  real X;  array H;
comment  This procedure evaluates the complex valued hankel
  function of the first kind for real argument X and integral order
  N and assigns it to H. The individual Bessel- and Neuman-func-
  tion series are not evaluated separately. Both the real and
  imaginary parts are generated from the same terms;
begin  real K, P, R, A, S, T, D, L;  integer Q;
  A := R := 1;  H[1] := H[2] := S := 0;
  for Q := 1 step 1 until N do begin R := R × Q;  S := S +
    1/Q end;  D := R/N;
  R := 1/R;  K := X × X/4;  P := (X/2)↑N;  T := ln(K) +
    1.1544313298631;
  for Q := 0, Q+1 while Q≦N∨L≠H[2] do
  begin  L := H[2];  H[1] := H[1] + A×K×R;
    H[2] := H[2] + A×(R×K×(T−S) − (if Q<N then D/P
      else 0));
    A := A×K/Q;  R := −R/(Q+N);  S := S+1/Q + 1/(Q+N);
    if Q<N then D := D/(N−Q)
  end;  H[2] := H[2] × .31830989
end
```

Notes on Programming Languages

On the Nonexistence of a Phrase Structure Grammar for ALGOL 60

Robert W. Floyd
Computer Associates, Inc., Woburn, Massachusetts

ALGOL 60 is defined partly by formal mechanisms of phrase structure grammar, partly by informally stated restrictions. It is shown that no formal mechanisms of the type used are sufficient to define ALGOL 60.

Let a phrase structure grammar be defined as a set of definitions in the Backus notation used to define ALGOL 60 [1]. A phrase structure language, then, is a language defined by such a grammar. In such a language, because of the finite number of syntactic types, all sufficiently long programs (blocks) contain a substring which in turn contains a proper substring of the same syntactic type as itself [2, 3] Thus, the program P takes the form $QRSTU$, where RST and S are of the same syntactic type so that either may be substituted for the other, and S is a proper substring of RST. Now $QR^{(i)}ST^{(i)}U$ is a syntactically correct program for any non-negative integer i, where $R^{(i)}$ denotes i occurrences of the string R.

Example. An ALGOL 60 program might contain the primary $(c \times d)$, which in turn contains the primary c. It would be possible, therefore, to replace $(c \times d)$ by c at that point in the program, or to replace c by $(c \times d)$ obtaining $((c \times d) \times d)$, etc., without destroying the syntactic correctness of the program.

The goal of the present paper is to exhibit a set of ALGOL 60 programs of unbounded length, and show that none of them has the property described by the first paragraph. This implies that ALGOL 60 is not definable by a phrase structure grammar alone.

Consider the ALGOL 60 program

begin real $x^{(n)}$; $x^{(n)} := x^{(n)}$ **end**

where $x^{(n)}$ stands for n occurrences of the letter x. If ALGOL 60 is a phrase structure language, we may choose n sufficiently large to make applicable the result of the first paragraph. That is,

begin real $x^{(n)}$; $x^{(n)} := x^{(n)}$ **end**

takes the form $QRSTU$, and $QR^{(i)}ST^{(i)}U$ is a syntactically correct program P_i for all $i \geqq 0$. A block in ALGOL must contain at least one declarator, a semicolon, and the words **begin** and **end**; since $QSU = P_0$ is a block, R and T can contain only the characters x and :=. Since the declarator **real** occurs only once and there are no commas, only one identifier is declared in each of P_i, and only that identifier may be used in P_i. Two cases arise:

(1) Neither R nor T contains :=. Then $R = x^{(j)}$ and $T = x^{(k)}$ with j and k not both zero. One cannot, however, delete x's from P_1 in two places R and T and still have all identifiers properly declared in P_0; at least three deletions would have to be made.

(2) R or T contains :=. Since P_0 does not contain :=, it must take the form

begin real $x^{(j)}$; **end**

Since R and T consist of x's with one occurrence of :=, P_i for $i > 0$ must take the form

begin real $x^{(f(i))}$; $x^{(f(i))} := x^{(f(i))} := \cdots := x^{(f(i))}$ **end**

containing i occurrences of :=, $i+2$ occurrences of $x^{(f(i))}$, and therefore $(i+2)(f(i))$ occurrences of x. Since the number of occurrences of x is a linear function $a + bi$ of i, we have

$$f(i) = \frac{a + bi}{2 + i} = \frac{a - 2b + b(2 + i)}{2 + i} = b + \frac{a - 2b}{2 + i}$$

always integer-valued. Then $a - 2b$ is zero, so $f(i) = b$; and the number of occurrences of x in P_i for all $i \geqq 0$ is $(i + 2)b$. Then P_1 is

begin real $x^{(b)}$; $x^{(b)} := x^{(b)}$ **end**

and P_0 is

begin real $x^{(2b)}$; **end**

which is not a subsequence of P_1 and cannot be obtained from P_1 by deletions.

The conclusion to be drawn is that it is not possible to state the formation rules of ALGOL 60 as a phrase structure grammar, so that there must necessarily be syntactic rules stated in other ways. The principal examples are the rules requiring the declaration of all variables, procedures, arrays and switches. It seems likely that similar considerations would apply to any other reasonable language in which all variables must be declared.

REFERENCES

1. NAUR, PETER (Ed.) ET AL. Report on the algorithmic language ALGOL 60. *Comm. ACM 3*, 5 (1960), 299–314.
2. CHOMSKY, N. On certain formal properties of grammars. *Information and Control 2* (1959), 137–167; A note on phrase structure grammars. *Information and Control 2* (1959), 393–395.
3. BAR-HILLEL, Y., PERLES, M., AND SHAMIR, E. On formal properties of simple phrase structure grammars. *Zeit. Phonetik, Sprachwissenschaft und Kommunikationsforschung 14* (1961), 143–172.

TALL—A List Processor for the Philco 2000 Computer

Julian Feldman

System Development Corporation, Santa Monica, California

Several of the computer languages that are oriented toward problems in symbol manipulation use a list type of memory organization.[1] The advantages of such a memory organization have been discussed elsewhere and will not be repeated here. The purpose of this note is to describe the method used in realizing a list language on the Philco 2000.

Information Processing Language V (IPL-V) was chosen as the source language for the list processor for the 2000 because this language has been well documented and has been implemented on several computers.[2] Heretofore, IPL-V has been implemented as an interpretive system. The interpretive system has three major components: (1) a loader which translates card images into internal machine words; (2) an interpreter which decodes instructions; and (3) a set of primitive processes, the "J's," which make up the bulk of the instruction vocabulary. The implementation of such an interpretive system has been a rather lengthy procedure usually estimated as taking six man-months.

IPL-V has been implemented on the 2000 as a set of macro-operations, subroutines and conventions supplementing TAC (*T*ranslator-*A*ssembler-*C*ompiler, the assembly language for the 2000).[3] These macro's, subroutines and conventions will be referred to as TALL (*TAC Li*st *L*anguage). TALL uses the loading facilities of TAC, the IPL-V primitive processes, and a set of subroutines performing the work of the interpreter. The macros aid in the translation from IPL-V to TAC. The macros and the primitive processes, the J's, can be placed on the TAC subroutine library tape and called in as required during assembly.

The implementation of IPL-V in this fashion has several advantages: (1) the time required to get a basic IPL-V system running on the 2000 was only three man-weeks; (2) symbolic machine language instructions can easily be inserted into TALL programs; (3) IPL-V statements can be used in conjunction with FORTRAN statements or JOVIAL statements;[4] and (4) no additional work is required to make TALL compatible with any monitor system for the 2000. A brief description of the TALL representations of IPL-V program and data follows.

TALL Program

The IPL-V program word has the format

P Q SYMB LINK

where P is an octal digit representing an operation code, Q is an octal digit specifying the degree of indirection represented by SYMB, SYMB is a machine address, and LINK is the machine address of the next instruction. In the TALL system, the P-Q combinations are represented as macro-operations which have SYMB and LINK as inputs. Thus the IPL-V program word is represented by the following line of TAC code:

L COMMAND ADDRESS

PQnn SYMB; LINK

The macro PQnn expands this line of code into two computer words. The first word has SYMB in the address of the left half-word and LINK in the address of the right half-word. The second word has a left half-word instruction which loads the first word into the A-register and a right half-word instruction which transfers to the subroutine PQnnX which finds its input parameters, SYMB and LINK, in the A-register. The conversion of program from IPL-V format to TALL format is a rather simple and straightforward procedure that can easily be accomplished by EAM equipment (an example is provided in the Appendix).

TALL Data

The IPL-V data word takes on various forms. The format for IPL-V symbolic data is the same as the format for program. The TALL format for symbolic data is the same as the program format with the exception that a "D" is added after the "PQnn."

---

[1] Program and Preprints of the ACM Conference on Symbol Manipulation. *Comm. ACM 3*, 4 (1960).

[2] NEWELL, A., ET AL. *Information processing language V manual*. Englewood Cliffs, Prentice-Hall, 1961.

[3] Philco 2000 TAC Manual. Philco Corp., Computer Div., Willow Grove, Penn., May 1961.

[4] Philco 2000 ALTAC Manual. Philco Corp., Computer Div., Willow Grove, Penn., Feb. 1961; C. J. SHAW, JOVIAL Manual. TM-555, System Development Corp., Santa Monica, Calif., 1961.